supposed to be entered in the journal. These 'Distracters' are included because sometimes errant documents occur in real life. The designer then uses the Domain Model features in the Knowledge Workbench to paint a Journal. An entity is created in the Domain Model to represent each transaction and each source document. Based on the twenty-two documents that the designer chose, she can anticipate errors that the student might make. For these errors, she creates topics of feedback and populates them with text. She also creates topics of feedback to tell the student when they have succeeded. Feedback Topics are created to handle a variety of situations that the student may cause.

The next step is to create profiles that the will trigger the topics in the concept tree (this task is not computational in nature, so the Transformation Component does not need to be configured) . A profile resolves to true when its conditions are met by the student's work. Each profile that resolves to true triggers a topic. To do some preliminary testing on the design, the designer invokes the Student Simulator Test Workbench. The designer can manipulate the Domain Model as if she were the student working in the interface. She drags accounts around to different transactions, indicating how she would like them journalized. She also enters the dollar amounts that she would like to debit or credit each account. She submits her actions to the component engines to see the feedback the student would get if he had performed the activity in the same way. All of this occurs in the test bench without an application interface. The last step in this phase is low-fi user testing. A test student interacts with a PowerPoint slide or bitmap of the proposed application interface for the Journalization Task. A facilitator mimics his actions in the test bench and tells him what the feedback would be. This simplifies low-fi user testing and helps the designer to identify usability issues earlier in the design when they are much cheaper to resolve.

Figures **14** and **15** illustrate the steps associated with a **build scenario** in accordance with a preferred embodiment. The instructional designer completes the initial interaction and interface designs as seen in the previous Scenario. After low-fi user testing, the Build Phase begins. Graphic artists use the designs to create the bitmaps that will make up the interface. These include bitmaps for the buttons, tabs, and transactions, as well as all the other screen widgets. The developer builds the interface using the bitmaps and adds the functionality that notifies the Domain Model of student actions. Standard event-driven programming techniques are used to create code that will react to events in the interface during application execution and pass the appropriate information to the Domain Model. The developer does not need to have any deep knowledge about the content because she does not have to build any logic to support analysis of the student actions or feedback. The developer also codes the logic to rebuild the interface based on changes to the domain model. A few passes through these steps will typically be required to get the application communicating correctly with the components. The debug utilities and Regression Test Workbench streamline the process. After the application interface and component communication are functioning as designed, the task is migrated to Usability testing.

The **Test Scenario** demonstrates the cycle that the team goes through to test the application. It specifically addresses usability testing, but it is easy to see how the tools also benefit functional and cognition testing. Again, we will use the Journalization Task as an example. Figure **16** illustrates a test scenario in accordance with a preferred embodiment. The test students work through the journalization activity. One of the students has made it over half way through the task and has just attempted to journalize the sixteenth transaction. The student submits to the Financial Coach, but the feedback comes back blank. The student notifies the facilitator who right-clicks on the Financial Coach's face in the feedback window. A dialog pops up that shows this is the twenty-seventh submission and shows some other details about the submission. The facilitator (or even the student in recent efforts) enters a text description of the problem, and fills out some other fields to indicate the nature and severity of the problem. All the student's work and the feedback they got for the twenty-seven submissions is posted to the User Acceptance Test (UAT) archive database. The instructional designer can review all the student histories in the UAT database

and retrieve the session where the student in question attempted the Journalization Task The designer then recreates the problem by replaying the student's twenty-seven submissions through the component engines using the Regression Test Workbench. The designer can then browse through each submission that the student made and view the work that the student did on the submission, the feedback the student got, and the facilitator comments, if any. Now the designer can use the debugging tools to determine the source of the problem. In a few minutes, she is able to determine that additional profiles and topics are needed to address the specific combinations of mistakes the student made. She uses the Knowledge Workbench to design the new profiles and topics. She also adds a placeholder and a script for a video war story that supports the learning under these circumstances. The designer saves the new design of the task and reruns the Regression Test Workbench on the student's session with the new task design. After she is satisfied that the new profiles, topics, and war stories are giving the desired coverage, she ships the new task design file to user testing and it's rolled out to all of the users.

Execution Scenario: Student Administration - Figure **17** illustrates how the tool suite supports student administration in accordance with a preferred embodiment. When a student first enters a course she performs a pre-test of his financial skills and fills out an information sheet about his job role, level, etc. This information is reported to the Domain Model. The Profiling Component analyzes the pre-test, information sheet, and any other data to determine the specific learning needs of this student. A curriculum is dynamically configured from the Task Library for this student. The application configures its main navigational interface (if the app has one) to indicate that this student needs to learn Journalization, among other things. As the student progresses through the course, his performance indicates that his proficiency is growing more rapidly in some areas than in others. Based on this finding, his curriculum is altered to give him additional Tasks that will help him master the content he is having trouble with. Also, Tasks may be removed where he has demonstrated proficiency. While the student is performing the work in the Tasks, every action he takes, the feedback he gets, and any other indicators of performance are tracked in the Student Tracking Database. Periodically, part or all of the tracked data are transmitted to a central location. The data can be used to verify that the student completed all of the work, and it can be further analyzed to measure his degree of mastery of the content.

Execution Scenario: Student Interaction - Figure **18** illustrates a suite to support a student interaction in accordance with a preferred embodiment. In this task the student is trying to journalize invoices. He sees a chart of accounts, an invoice, and the journal entry for each invoice. He journalizes a transaction by dragging and dropping an account from the chart of accounts onto the 'Debits' or the 'Credits' line of the journal entry and entering the dollar amount of the debit or credit. He does this for each transaction. As the student interacts with the interface, all actions are reported to and recorded in the Domain Model. The Domain Model has a meta-model describing a transaction, its data, and what information a journal entry contains. The actions of the student populates the entities in the domain model with the appropriate information. When the student is ready, he submits the work to a simulated team member for review. This submission triggers the Analysis-Interpretation cycle. The Transformation Component is invoked and performs additional calculations on the data in the Domain Model, perhaps determining that Debits and Credits are unbalanced for a given journal entry. The Profiling Component can then perform rule-based pattern matching on the Domain Model, examining both the student actions and results of any Transformation Component analysis. Some of the profiles fire as they identify the mistakes and correct answers the student has given. Any profiles that fire activate topics in the Remediation Component. After the Profiling Component completes, the Remediation Component is invoked. The remediation algorithm searches the active topics in the tree of concepts to determine the best set of topics to deliver. This set may contain text, video, audio, URLs, even actions that manipulate the Domain Model. It is then assembled into prose-like paragraphs of text and media and presented to the student. The text feedback helps the student localize his journalization errors and understand

why they are wrong and what is needed to correct the mistakes. The student is presented with the opportunity to view a video war story about the tax and legal consequences that arise from incorrect journalization. He is also presented with links to the reference materials that describe the fundamentals of journalization. The Analysis-Interpretation cycle ends when any coach items that result in updates to the Domain Model have been posted and the interface is redrawn to represent the new domain data. In this case, the designer chose to highlight with a red check the transactions that the student journalized incorrectly.

The Functional Definition of the ICAT

This section describes the feedback processes in accordance with a preferred embodiment. For each process, there is a definition of the process and a high-level description of the knowledge model. This definition is intended to give the reader a baseline understanding of some of the key components/objects in the model, so that he can proceed with the remaining sections of this paper. Refer to the Detailed Components of the ICAT for a more detailed description of each of the components within each knowledge model. To gain a general understanding of the ICAT, read only the general descriptions. To understand the ICAT deeply, read this section and the detailed component section regarding knowledge models and algorithms. These processes and algorithms embody the feedback model in the ICAT. There are six main processes in the ICAT, described below and in more detail on the following pages.

Figure **19** illustrates the remediation process in accordance with a preferred embodiment. Remediation starts as students interact with the application's interface (process #1). As the student tries to complete the business deliverable, the application sends messages to the ICAT about each action taken (process #2). When the student is done and submits work for review, the ICAT compares how the student completed the activity with how the designer stated the activity should be completed (this is called domain knowledge). From this comparison, the ICAT get a count of how many items are right, wrong or irrelevant (process #3). With the count complete, the ICAT tries to fire all rules (process #4). Any rules which fire activate a coach topic (process #5). The feedback algorithm selects pieces of feedback to show and composes them into coherent paragraphs of text (process #6). Finally, as part of creating feedback text paragraphs, the ICAT replaces all variables in the feedback with specifics from the student's work. This gives the feedback even more specificity, so that it is truly customized to each student's actions.

Knowledge Model - Interface Objects In any GBS Task, the student must manipulate controls on the application interface to complete the required deliverables. Figure **20** illustrates the objects for the journalization task in accordance with a preferred embodiment. The following abstract objects are used to model all the various types of interface interactions. A SourceItem is an object the student uses to complete a task. In the journalization example, the student makes a debit and credit for each transaction. The student has a finite set of accounts with which to respond for each transaction. Each account that appears in the interface has a corresponding SourceItem object. In other words, the items the student can manipulate to complete the task (account names) are called SourceItems. A Source is an object that groups a set of SourceItem objects together. Source objects have a One-To-Many relationship with SourceItem objects. In the journalization example, there are four types of accounts: Assets, Liabilities and Equity, Revenues, and Expenses. Each Account is of one and only one of these types and thus appears only under the appropriate tab. For each of the Account type tabs, there is a corresponding Source Object. A Target is a fixed place where students place SourceItems to complete a task. In the journalization example, the student places accounts on two possible targets: debits and credits. The top two lines of the journal entry control are Debit targets and the bottom two lines are Credit targets. These two targets are specific to the twelfth transaction. A TargetPage is an object that groups a set of Target objects together. TargetPage objects have a One-To-Many relationship with Target objects (just like the Source to SourceItem relationship). In the journalization example, there is one journal entry for each of the twenty-two